Project information

Created on
February 16, 2026

Name	Last update
backend	1 day ago
frontend/VotingSystem	1 day ago
load-test	6 days ago
.env-example	6 days ago
.gitignore	2 months ago
README.md	6 days ago
docker-compose.dev.yml	1 day ago
docker-compose.yml	6 days ago
traefik-dynamic.yml	3 weeks ago
traefik-static.yml	3 weeks ago
votingsystem.sln	2 months ago

README.md

VotingSystem

VotingSystem is a full-stack web application with a React/Vite frontend and a .NET 10 Web API backend, backed by PostgreSQL. The application demonstrates a scalable, load-balanced architecture using Docker Compose and Traefik for reverse proxy and routing.

Project Structure

```
.
├── backend/                  # .NET 10 Web API
│   └── VotingSystem.Api/     # Main API project
├── frontend/
│   └── VotingSystem/         # React + Vite frontend
├── docker-compose.yml        # Production containers
├── traefik-static.yml        # Traefik configuration
├── traefik-dynamic.yml       # Dynamic routing rules
└── load-test/                # Load testing scripts
```

Technology Stack

Component	Technology	Version
Backend Language	.NET	10.0
Backend Framework	ASP.NET Core	10.0

Component	Technology	Version
Backend ORM	Entity Framework Core	10.0.3
Database	PostgreSQL	18.3
Frontend Runtime	Node.js	24.14.1-alpine
Frontend Library	React	19.2.0
Frontend Build Tool	Vite	7.3.1
Frontend Language	TypeScript	~5.9.3
Frontend State Management	Zustand	5.0.12
Reverse Proxy & Load Balancer	Traefik	v3.6.13
Frontend Server	Nginx	1.28.3-alpine
Containerization	Docker & Docker Compose	Latest

Prerequisites

For Docker-based Deployment

- **Docker** - Download
- **Docker Compose** - Included with Docker Desktop

Running the Project

Production Deployment with Docker Compose

The entire application stack can be started with a single command:

```
docker compose up --build
```

This will start:

- **Traefik** (Reverse Proxy): http://localhost (main app) and http://localhost:8080 (dashboard)
- **Backend API** (2 load-balanced instances): Internal service listening on port 8080
- **Frontend** (Nginx): Served through Traefik at http://localhost
- **PostgreSQL**: Internal service with persistent data volume

Application Access

Once running:

- **Frontend**: http://localhost
- **Traefik Dashboard**: http://localhost:8080

Shutdown

To stop the containers:

```
docker compose down
```

To also remove volumes (database data):

```
docker compose down -v
```

Architecture Overview

Request Routing with Traefik

The application uses Traefik v3 as a reverse proxy to route traffic:

- **Frontend Routes**: `http://localhost/` → Nginx SPA server

- **API Routes**: `http://localhost/api/*` → Backend service (load balanced across 2 instances)
  - Traefik automatically strips the `/api` prefix before forwarding
  - Health checks performed every 10 seconds on `/api/health`
  - Automatic failover if an instance becomes unhealthy

Backend Features

- **JWT Authentication**: 20-minute token expiration with configurable issuer/audience + 7-day refresh token
- **Automatic Migrations**: Entity Framework Core migrations run automatically on startup with retry logic
- **Health Checks**: Dedicated health endpoint for container orchestration
- **CORS**: Configured to allow cross-origin requests

Frontend Features

- **Single Page Application (SPA)**: All routes served via Nginx with proper SPA routing
- **Static Asset Caching**: Long-term caching for immutable assets (1 year)
- **TypeScript**: Full type safety across the application
- **State Management**: Zustand for lightweight client-side state

Performance Metrics

Load test results using `wrk` (WebSocket benchmarking tool):

```
✅ Load Test Results: GET /api/votes/my-votes (Protected Endpoint)
═══════════════════════════════════════════════════════════════

Running 1m test @ http://localhost:80/
  4 threads and 50 connections

Thread Stats   Avg      Stdev     Max   +/- Stdev
  Latency    44.84ms  132.65ms   1.12s    95.40%
  Req/Sec   701.23    120.76     0.90k    84.18%

168796 requests in 1.07m, 235.77MB read
Requests/sec:   2638.92
Transfer/sec:      3.69MB

Errors:         0 ✅
═══════════════════════════════════════════════════════════════
```

Key Metrics:

- **Throughput**: 2,638.92 requests/second
- **Total Requests**: 168,796 in 60 seconds
- **Avg Latency**: 44.84ms
- **Error Rate**: 0%
- **Data Transfer**: 235.77MB

Configuration & Environment Variables

Frontend and Backend

Key environment variables that can be set in .env file:

```
DB_NAME=your-db-name
DB_USER=your-db-username
DB_PASSWORD=your-db-password
ASPNETCORE_ENVIRONMENT=your-backend-environment
VITE_API_URL=your-vite-url
JWT_SECRET=your-jwt-secret
```

Database

- **Host**: postgres (internal container)
- **Port**: 5432
- **Database**: your-db-name
- **User**: your-db-username

- **Password**: your-db-password
- **Persistence**: Volume mounted for data persistence across restarts

Maintenance

- **Automatic Migrations**: The backend automatically runs Entity Framework Core migrations on startup. Ensure database backups exist before deployments.
- **Health Checks**: Traefik monitors backend health via the `/api/health` endpoint. Failed health checks result in automatic traffic rerouting.

Monitoring

Traefik Dashboard

Access the Traefik dashboard while containers are running:

```
http://localhost:8080
```

The dashboard shows:

- Active services and their status
- Request statistics
- Routing rules
- Health check status

Load Testing

For detailed load testing instructions, see the Load Test README.

The load-test folder contains:

- Setup scripts to generate test data and user tokens
- Load test scenarios (baseline, standard, heavy load)
- `wrk` Lua scripts for executing protected endpoint tests
- Prerequisites and step-by-step execution guides

Quick start:

1. Ensure the application is running: `docker compose up`
2. Navigate to the load-test directory: `cd load-test`
3. Generate test data: `./setup-test-data.sh http://localhost:80 10 3`
4. Run a load test: `wrk -d 60 -c 50 -t 4 -s load-test.lua http://localhost:80/`

Results will show throughput, latency, and error rates for protected endpoints using JWT authentication.

Additional Resources

- .NET Documentation
- React Documentation
- Vite Documentation
- Docker Documentation
- PostgreSQL Documentation
- Traefik Documentation